06 | 通信设计：请不要让消息通信拖垮了系统的整体性能

2021-05-29 尉刚强

《性能优化高手课》 课程介绍 >

讲述：尉刚强
时长 18:39 大小 17.08M

你好，我是尉刚强。随着业务规模的不断扩大，分布式的系统架构越来越流行，而基于消息队列的通信设计，则是分布式系统性能设计中非常关键的一环，今天我们就来聊一聊这个话题。

首先，为什么基于消息队列的通信设计如此重要呢？其实简单来说，它在软件系统中的地位和作用，类似于接力赛中的交棒环节，一旦某个选手在交接棒期间失误，那么整个团队的成绩就会被拖垮。所以，如果没有做好基于消息队列的通信设计，那么系统的整体性能就一定不会很理想。

> 之前我做过一个项目，在业务系统中大量使用了 Redis 来作为消息队列进行传输，而业务开发人员经常就会因为接口使用不当，导致消息丢失；另外，由于队列容量也不加以

> 控制，并会出现消息队列的数据堆积，使得整个 Redis 的速度都很慢，进而就导致了业务整体的性能非常不稳定。

另外，还有一个很重要的原因，那就是基于消息队列的通信设计，与软件系统的很多设计都有关系，比如并发设计、IO 设计等，它对整个软件架构的影响比较大。所以在软件的设计阶段，我们就应该开始关注它。

今天这节课上，我会给你详细介绍基于消息队列选型和设计过程中的一些核心要点，以及它们对软件性能的影响。我会带你理解正确开展通信队列的设计思路，从而让你真正掌握分布式系统中进行通信设计的方法，避免因消息通信的时延与吞吐量制约你的软件性能的提升。

那么具体是什么思路呢？在对一个高性能分布式系统进行通信队列设计时，我一般会按照这几个步骤开展设计工作，分别是消息队列选型设计、消息体设计、消息通信过程设计。因为消息队列选型是最基础的，它会直接影响后续的通信过程设计。

所以接下来，我们就从通信队列的选型设计开始学习吧。

消息队列选型设计

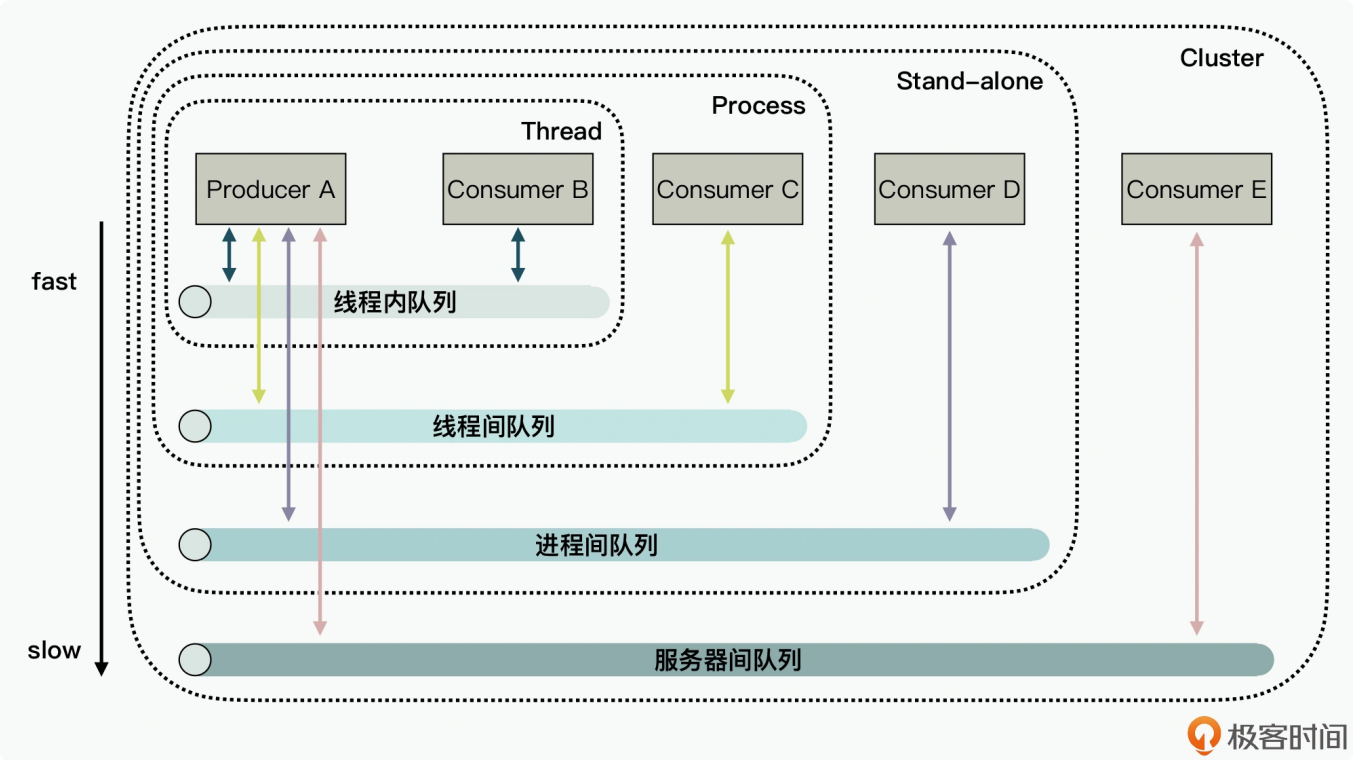
首先，在软件设计阶段，消息队列选型设计的核心目的，其实就是根据软件架构选择合适的消息队列底层实现、对应的框架或服务等，从而将底层的传输成本降低到最小化。而且现在是一个富技术的时代，碰到问题时我们可以选择的框架和服务特别多，而前面我也说过，**具体的消息队列的实现原理及性能，会制约软件的架构设计。**

所以，如果说消息队列选型失败，那么后续即使你花费再大的代价，可能都无法弥补性能上的不足。我曾经就经历过一个项目，该项目对时延要求非常高，可是因为选用了 RabbitMQ 这款消息队列系统，它在传输消息时，都需要多经过一个代理服务器，比在实例间直接点对点通信要慢很多，所以最后就无法满足系统延迟的极致需求。

可见，我们在进行消息队列选型设计之前，**一定要确认好需要进行通信的软件实例的运行位置**，避免由于搞不清楚该选择什么样的底层消息传输机制，影响到最后的软件性能。

那接下来你可能要问：具体该怎么根据不同软件实例的位置分布，来选择不同的底层消息传输机制呢？

现在我们来看一张图片，这是一张不同层级的消息队列底层原理图，图中按照底层实现机制，将消息队列分为了四个层级，分别是线程内队列、线程间队列、进程间队列、服务器间队列。这四个层级从上到下，其传输时延会越来越大。



这也就是说，如果我们在线程内通信选择了线程间队列，虽然它也可以满足功能需求，但是由于需要解决同步互斥的问题，其性能往往不会是最佳的。所以，**在选型设计阶段，我们要优先选择能够满足通信要求的同时，底层实现也是最快的消息队列类型。**

不过，在实际的系统设计和实现过程中，很多人其实都忽视了不同底层实现的消息队列之间，所具备的性能差异。所以接下来，我就给你一一介绍下前面这四种消息队列类型的实现原理，以及在做选型设计时的核心关注点。

不同类型的消息队列都有啥特点？

首先是**线程内队列**。现在我们已经知道，线程内队列的传输速度最快、时延最低，它可以作为同一个线程的各个模块单元间通信，因为不用考虑并发问题，所以性能会比较高。

其次是**线程间队列**。在一个进程中，我们可以基于内存创建队列来进行通信，但是由于跨线程内存访问数据一致性的问题，很容易引起执行结果的不确定性，所以这时我们就需要考虑前面第 3 讲介绍过的同步互斥问题。也就是说，在使用线程间队列的过程中，可能会引入加锁或内存屏障机制，因而就会导致性能下降。

第三种是**进程间队列**。我们可以利用 IPC（Inter-Process Communication，进程间通信）队列，或者利用进程间共享的一块内存建立通信队列，因为这种通信队列可以完全在内存中实现，所以几乎可以达到与线程间队列比较接近的性能。比如，Java 语言的 traffic-shm(Anna) 库提供的消息队列，由于不需要跨服务器的消息队列，从而可以避免引入额外的网络传输的开销。

最后是**服务器间队列**。它的实现原理是基于物理网络设备，建立传输网络来进行通信，所以这样可能就会受到传输带宽、网络拥塞等各种问题的影响，传输时延相对会比较长，而且有可能不稳定。

实际上，线程内、线程间、进程间通信队列设计在嵌入式分布式系统中使用得很频繁，而对于网络服务化的分布式系统而言，这些通信队列机制通常已经内置到了特定语言或者框架库中，比如 Go 语言的 Channel 队列、Java 的 CurrentQueue 等。

不过，**对于互联网的分布式系统来说，消息队列选型设计的核心关注点，主要在于服务器间的消息队列选型。**因为大部分的软件 / 服务实例都部署在不同的机器之上，所以我们只能使用服务器间消息队列。

那么针对服务器间的消息队列，我们可以选择采用的消息队列实现非常多，比如有 RabbitMQ、ActiveMQ、Kafka、RocketMQ、ZeroMQ，等等。当然，还有一些基于特定数据库封装实现的消息队列，如基于 Redis 实现的消息队列等。

既然如此，**在这些消息队列之间，我们又应该怎么选择呢，它们对性能都有什么影响呢？**接下来，我就给你分享一下选择满足业务性能需求的消息队列的方法。

如何选择满足业务性能需求的消息队列？

首先你要知道的是，虽然前面我列举的这些消息队列都可以实现不同服务器间的消息通信，但因为它们在底层实现上存在差异，会直接影响软件在性能上的表现，我给你举几个例子。

比如，对于 ZeroMQ 来说，它是基于 C 语言开发的，并没有中间代理服务器来缓存消息，会直接基于服务器中间的网络链路进行通信，所以它的时延速度是最高的。

而 Kafka 是一款多分区、多副本，且基于 ZooKeeper 协调的分布式消息系统，理论上可以支撑非常大的集群规模，所以它可以支撑的业务吞吐量会非常高。我之前在开发大数据平台时，利用 Kafka，使得消息吞吐量可以支撑到几百 MB/s 的速度。

那么，相比 Kafka 而言，RabbitMQ 的吞吐量会差一些，但是它在时延性能和功能上并不逊色，可以满足大部分业务场景的性能需求。

再举个例子，我看到有很多的软件系统，其实都是使用数据库来构造消息队列的，比如基于 Redis 开发的消息队列等。在理论上，你确实是可以基于各种分布式数据库来开发消息队列的。但是这样的做法，一方面会受到不同数据库实现架构的影响，另一方面它也不具备很多针对消息通信的优化设计与开发配置策略，所以往往在性能上不会是最佳的。

所以综合以上不同消息队列的特性和使用场景，我想告诉你的是，**当你在面对一些消息通信很关键的性能场景时，针对消息队列的选型，需要从时延、吞吐量等多个维度进行性能评估与分析，而在评估分析前，基于底层实现机制进行选型是你要做的第一步。**

好了，在确认了消息队列选型之后，为了支撑分布式系统通信的高性能，你还需要做好消息体设计，下面我们就具体来看看它的核心关注点。

消息体设计

实际上，消息体设计的核心有两点，消息内容设计和消息编码设计。

首先，什么是消息内容设计呢？是这样的，同样一个信息，通常会有很多种表示方法。比如“北京”和 “中国的首都城市” ，它们都表示相同的地方，但是二者所需要的数据量（字数）却是不一样的。

也就是说，**不同的消息内容呈现方式会直接影响传输的消息体大小**，这对通信性能很关键，但是这部分却经常被我们所忽略。

而除了消息内容设计之外，不同的消息编码格式设计也会影响传输的消息体大小，从而直接影响到消息传输的时延和吞吐量。现在比较常见的消息编码格式主要有几种：TLV 格式（Type/Tag、Length、Value）、ProtoBuffer 格式、JSON 格式、XML 格式。

其中，**TLV 格式和 ProtoBuffer 格式并不是自描述的**，因此就需要生产者和消费者之间提前约定好消息格式，或者基于特定的格式描述文件进行解释；而**使用 JSON 和 XML 这类消息格式，通常消息内容是自描述的**，所以消息体中会携带额外的描述消息，进而就造成消息体比较大，因此我不太建议在一些高性能的通信业务场景中使用。

所以说，如果消息通信性能对你的软件系统性能很关键的话，你就应该优先考虑前两种消息编码格式。但是采用这两种方式，你还需要在软件实现中处理接口的兼容性，因而会花费更多的精力。

事实上，针对关系型数据库，**表信息**就是对字段内容格式的描述，所以在一些特殊场景下，我们基于数据库也可以构造出一些高性能的通信队列。比如，在 Mongo 的主节点和其他 Replica 节点之间，传输信息所使用的 Change Stream 就是使用数据库内一个 Collection（集合）来实现的，因此你也可以构造出一些高性能通信队列的场景。

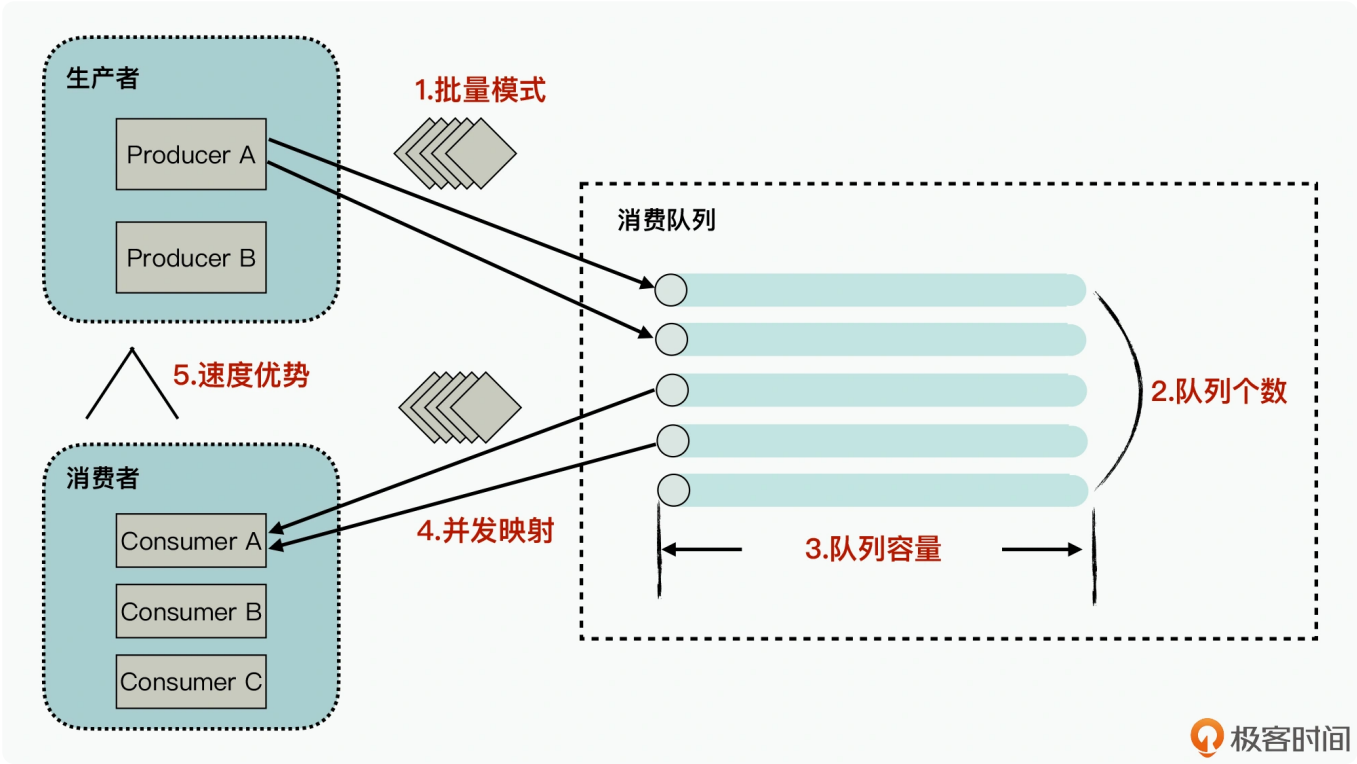
而当你使用后面两种消息格式时，那么在发送和接收消息的过程中，你可以使用一些**编解码库**来压缩消息体的内容，从而减少传输的信息大小。比如针对 JSON，你可以使用 cJSON、HPACK 这类的压缩算法来压缩传输数据，通常压缩效率还是比较高的。

OK，在选择完合适的通信队列实现，并完成了消息的高效编码之后，是不是就可以保证通信性能卓越，不会影响系统业务的性能呢？

当然不是的，**配置和使用消息队列以及匹配软件的设计，对性能的影响同样也非常大**。所以接下来，我们就一起看看，如何在通信设计的过程中保证系统的高性能。

消息通信过程设计

首先，我们来看一张图片，它展示的就是一个基于消息队列通信的过程，图上包含了三个核心概念，分别是生产者、消费者和消息队列：



其中，生产者和消费者都可以包含多个运行实例，而消息队列则是消息传输通道的一个抽象载体，在有些消息队列服务框架或服务中，如 Kafka、RabbitMQ 等，会有独立的运行实体去处理它。但是在一些场景中，消息队列并没有独立的运行实体可被处理，而是依托在生产者或消费者进程中进行处理的。

另外在图上，我还重点标注了对性能影响比较关键的几个决策点。**理解了这些核心决策点和背后的原理后，你就可以针对分布式架构的通信部分，设计出高性能方案了**：

1. 批量模式：即从消息队列读取和写入消息的模式，我们应该优先选择批量或 Batch 模式。
2. 队列个数：即基于性能设计去选择消息队列的个数。
3. 队列容量：即消息队列中可以保存消息的个数或是字节的长度限制。
4. 并发映射：即在消息队列与业务架构中，生产者和消费者对于运行实例之间的映射关系设计。
5. 速度优势：即消费者处理消息的速度应大于生产者生产消息的速度。

那么我们该怎么理解以上这 5 个决策点呢？下面我就一一来给你详细介绍下。

首先是**批量模式**，通常对于消息队列而言，单个消息与批量消息的读取与写入的性能差异非常大，所以很多消息队列在这里都提供了很灵活的配置能力，比如 Kafka 客户端的 batch 大小和最大时延，或者 RabbitMQ 中接收端的 prefetch 等配置。也就是说，我们在业务代码中设计消息队列通信之前，一定要针对业务模型，调整和分析批处理模式中的相关配置。

同时，你还需要明白的一点是，即使业务中按照单条消息进行处理，而发送消息和接收消息也采用批量消息模式，这两个其实并不冲突。在上节课我也介绍过，从业务使用的视角来看，消息队列也属于 IO 交互，所以你还需要依据 IO 异步交互设计来进一步提升性能。

然后是**消息队列的个数**，对于拥有独立的 Broker 节点的消息队列服务来说，其实也是比较重要的。因为在消息队列服务的设计实现中，通常每个队列是由独立的线程来处理的。所以，提升消息队列个数，也就是提升消息通信中可以使用的 CPU 资源。比如，在 RabbitMQ 中，一个队列对应一个线程，它的上限吞吐量是确定的，所以你就可以通过提升消息队列个数，来提升消息传输的性能。

此外，**消息队列的容量设置**也是对性能影响比较关键的一个因素，但是却经常被我们所忽视。要知道，消息队列通常可以作为生产者与消费者中间的流控手段，当消费者处理能力下降时，我们可以通过消息队列的容量来限制生产者继续添加消息，从而限制生产者接收处理消息。

而如果这个容量设置过大的话，首先就会导致流控机制丧失，其次还有可能引起消息队列占用资源过多，影响整个系统的性能。其实，在一些高性能的嵌入式分布式系统中，消息队列长度都是需要进行严格设计的。

那并发映射主要解决的是什么问题呢？其实，它主要解决的是同步互斥的问题。

并发映射的关键，就是避免消费者或生产者的运行实例与消息队列之间的映射关系，出现同一个消息队列被多个运行实例并发访问，导致引入同步互斥的问题，尤其是对进程间的消息队列设计来说最为关键。

另外，针对跨服务间存在 Broker 节点的消息队列来说，我们也可以使用并发映射，同样可以减少 Broker 节点中处理消息队列的复杂度。

最后我要说的一点就是**速度优势**，在设计系统时，我们要尽量保证消费者的处理速度大于生成者，这是一个很重要的原则，因为它可以避免出现消息队列被阻塞的场景。但这里需要注意的是，处理速度大并不一定就是要求进程实体多，因为不同的业务逻辑，使用一个进程中的处理速度都是不一样的。

好了，除此之外，在消息通信过程设计中，其实还有一些对性能设计比较关键的点，比如持久化等，但这些点通常是我们已经能够关注到的点，所以这里我就不展开讲解了。

小结

今天这节课，我主要是从消息队列选型设计、消息格式编码设计、通信过程设计三个阶段，给你展开讲解了针对通信性能的关键技术手段。

当你面对一个高性能软件系统的通信设计时，你也可以按照今天我介绍的步骤，首先根据业务中并发实例的位置布局，选择和底层实现相匹配的消息队列；然后再根据业务逻辑特点，设计消息内容表示和编解码方式；最后，针对通信过程的 5 个核心关注点，有针对性地调整软件的设计与实现。这样，你就可以在使用同样的软硬件资源情况下，实现出更好的性能通信设计了。

思考题

我们在选用 RabbitMQ 进行软件通信设计的过程中，应该怎样在安全可靠性和性能之间做好平衡呢？

欢迎在留言区分享你的答案和思考。如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | IO设计：如何设计IO交互来提升系统性能？

下一篇 07 | 数据库选型：如何基于性能需求选择合适的数据库？

精选留言 (3)

写留言

Dream.
2021-06-10

消费速度要比生产快，基本上都是一些跨部门服务之间的服务通信，此时队列的用处就是用于业务解耦。

实际业务上，大多数情况都是因为响应慢，然后引入队列对业务进行进行异步处理...这种情况下引入的队列，正常情况下队列的消费都赶不上生产速度。解决这种情况，我能想...
展开

作者回复: 提升队列，主要解决消息通道的速度，是否提升消费处理速度，可能还依赖于业务具体实现。

danger boy
2021-06-10

根据业务需求和资源大小，选择合适的MQ软件
消息传递规，选择占用体积小的编码格式，尽量保证数据内容的精简

批量生产消费
消费速度大于生产速度，避免消息堆积

jjn0703
2021-05-30

“比如，对于 ZeroMQ 来说，它是基于 C 语言开发的，并没有中间代理服务器来缓存消息，会直接基于服务器中间的网络链路进行通信，所以它的时延速度是最高的。” 请问这句话的时延速度是什么概念呀？是消息传递时延最短的意思么？

作者回复: 嗯嗯，相比较其他通信队列技术而言，传输时延可以做的更低。